Just Ask!

Question: Recently we found a problem in some code we had that was using the naked indicator. This code `I ^(1)=^Y(2) ...` does not appear to work the same as `S ^(1)=^Y(2)`. Is this correct?

Editors: The use of the naked indicator is primarily a short-hand notation that is left over from some of the MUMPS dialects that preceded the current M language. These dialects worked on hardware with limited resources. Global buffers, as we currently understand them under most modern implementations, were limited to one buffer per process and contained the last physical block of global data accessed by the process.

In these dialects, a full global reference (`^TEST(A,B)`) forced the system to physically access the disk, while a naked reference (`^(B)`) told the system to look at the last block accessed for the date before going to the physical disk. These systems predated the use of B-trees for data storage. The person designing the system had to put a great deal of thought into the particular hashing algorithm(s) used for determining subscripts and what data were stored where.

If a global design had three levels of subscripts, then a full reference always caused at least three physical disk reads. On the other hand, if the data sought were not found in the first data block, then continuation blocks were used, another time-consuming process. The system designer's challenge was to find the best balance between subscript depth and continuation blocks to minimize the number of physical disk reads.

Once the database was designed, the programmer also could help reduce the number of physical reads by clustering the data accesses combined with use of the naked reference. A naked reference simply told the system to look at the last block accessed (kept in the disk buffer for the process) and look for the data there. If the particular node was not found in the disk buffer, the system constructed a full global reference and began searching for the node from the top of the global, just as it did for an explicate full reference.

This obsession with limiting disk access (and reducing the number of machine instructions) is evident in other characteristics in the M language. The evaluation of expressions is done from left to right without operator precedence to minimize stacking and look-ahead. The SET command does not look at the target storage location (the left side of the equal sign) until it is ready to store a value (the value of the right side of the equal sign).

Thus, the purpose of the naked indicator was performance. In modern M implementations, a naked reference has no performance advantage over an explicate full reference. In theory, the opposite may be true. A naked reference may require more processing while it reconstructs the full reference. Without any performance advantage, why was the naked reference carried over into modern M? The primary reason was to ease the conversion of MUMPS dialects to standard M. If the logical function of the naked syntax was the same between the dialect and the standard, the programmers would not have to spend as much time backtracking through the code to determine what the original programmer intended the naked reference to reference.

Today, the reason for using the naked reference has changed. Like most tools, the people (programmers) actually using the tool found another use for the naked: shorter code. Much like contractions, acronyms and pronouns are used in English, the naked indicator is used by programmers to say the same thing in less space. Like the English counterpart, understanding the actual intention sometimes takes a bit of work. The meaning of *he drove the bus* is clear only in context of what came before this sentence, sometimes way before. There could be three different interpretations: what the author intended, what a reader understands, and what strict parsing would produce.

In the examples above, `I ^(1)=^Y(2) ...` requires the evaluation of an expression, left to right. To determine what variable is used for the `^(1)` requires you to know what the last global reference was in the code preceding. That value is then compared with the value in `^Y(2)`.

In the SET statement, `S ^(1)=^Y(2)`, the value to be assigned, `^Y(2)`, is determined first, then M looks to see where to place the value. Since the location is a naked reference, it becomes dependent on the preceding global reference, the `^Y(2)`. Thus, the storage location is `^Y(1)`.

While getting naked may be fun and exciting, it can also lead to misunderstanding, confusion, and, ultimately, get you into trouble.

Send *Just Ask!* questions or requests to the managing editor at *M Computing*.